

LAPORAN PRAKTIKUM DATA STRUCTURE

Recursion

Dosen Pengampu:

H. Fatchurrochman,M.Kom

Asisten Praktikum:

Fillah Anjany 230605110033



Oleh :

Muhammad Alif Mujaddid

240605110082

Jurusan Teknik Informatika Fakultas Sains dan Teknologi

UIN Maulana Malik Ibrahim Malang

2025

A. PENDAHULUAN

1. Rekursi merupakan teknik dimana method/fungsi memanggil dirinya sendiri. Sebutkan minimal tiga contoh program yang dapat diimplementasikan menggunakan teknik rekursi! Untuk masing-masing contoh tersebut, jelaskan bagaimana penggunaan rekursi dalam menyelesaikan masalah pada contoh yang Anda sebutkan!

Jawab :

1. Faktorial

Program menghitung faktorial dari suatu bilangan n dapat diimplementasikan dengan rekursi. Secara matematis, faktorial didefinisikan sebagai:

$$n! = n \times (n-1)!$$

dengan kondisi dasar $0! = 1$

Dalam rekursi, fungsi faktorial(n) akan memanggil dirinya sendiri dengan parameter $n-1$ hingga mencapai nilai dasar 0 atau 1. Rekursi digunakan di sini karena masalah faktorial dapat dipecah menjadi submasalah yang lebih kecil dengan pola yang sama.

2. Fibonacci

Deret Fibonacci didefinisikan dengan rumus:

$$F(n) = F(n-1) + F(n-2)$$

dengan kondisi dasar $F(0) = 0$ dan $F(1) = 1$.

Dalam implementasinya, fungsi fibonacci(n) akan memanggil dirinya sendiri dua kali, yaitu fibonacci($n-1$) dan fibonacci($n-2$), hingga mencapai kondisi dasar. Penggunaan rekursi membantu memodelkan definisi matematis Fibonacci secara langsung, meskipun dalam praktiknya bisa dioptimalkan dengan metode iterasi atau memoization.

3. Pencarian Biner (Binary Search)

Binary search digunakan untuk mencari suatu elemen dalam array yang sudah terurut.

Ide utamanya adalah membagi array menjadi dua bagian pada setiap langkah:

- Bandingkan nilai tengah dengan nilai yang dicari.
- Jika sama, pencarian selesai.
- Jika nilai yang dicari lebih kecil, pencarian dilanjutkan ke bagian kiri.
- Jika lebih besar, pencarian dilanjutkan ke bagian kanan.

Dengan rekursi, fungsi binarySearch(arr, left, right, target) akan memanggil dirinya sendiri dengan batas indeks yang lebih kecil setiap kali sampai elemen ditemukan atau batas pencarian habis. Rekursi membantu menyelesaikan masalah ini karena proses pencarian selalu berulang dengan subset array yang lebih kecil, sesuai prinsip divide and conquer.

2. Menghitung bilangan segitiga (triangular number)



Rekursi dapat digunakan untuk menghitung bilangan segitiga tanpa menggunakan iterasi (for, while, atau do-while). Berikut ini listing program method perhitungan bilangan segitiga.

Iterasi	Rekursi
<pre>public int triangleIter(int n) { int result = 0; for (int i = n; i > 0; i--) { result += i; } return result; }</pre>	<pre>public int triangleRecur (int n) { if (n == 1) { return 1; } else { return n + triangleRecur (n - 1); } }</pre>

Lengkapi listing program tersebut dalam sebuah class dan tambahkan method main untuk memanggil masing-masing method tersebut. Jalankan program untuk menghitung bilangan segitiga dari

5. Tuliskan output program yang telah Anda lengkapi!

```
1 public class Triangle {
2     public int triangleIter(int n) {
3         int result = 0;
4         for (int i = n; i > 0; i--) {
5             result += i;
6         }
7         return result;
8     }
9     public int triangleRecur(int n) {
10        if (n == 1) {
11            return 1;
12        } else {
13            return n + triangleRecur(n - 1);
14        }
15    }
16 }
17
18 public static void main(String[] args) {
19     Triangle tn = new Triangle();
20
21     int n = 5;
22     System.out.println("Bilangan segitiga dari " + n + " (Iterasi): " + tn.triangleIter(n));
23     System.out.println("Bilangan segitiga dari " + n + " (Rekursi): " + tn.triangleRecur(n));
24 }
```

```
addid@LAPTOP-F80MDN28 MINGW64 /a/Code/DSA (main)
$ cd a:\\Code\\DSA ; /usr/bin/env A:\\Java\\jdk-22\\bin
orkspaceStorage\\df75c84a286eacae2dce0b54ebb13767\\redha
● Bilangan segitiga dari 5 (Iterasi): 15
  Bilangan segitiga dari 5 (Rekursi): 15
```

3. Pahami dan bandingkan langkah perhitungan bilangan segitiga antara menggunakan teknik perulangan dan dengan teknik rekursi. Tuliskan langkah perhitungan sesuai program nomor 3 yang telah Anda jalankan! Jelaskan!

Iterasi:

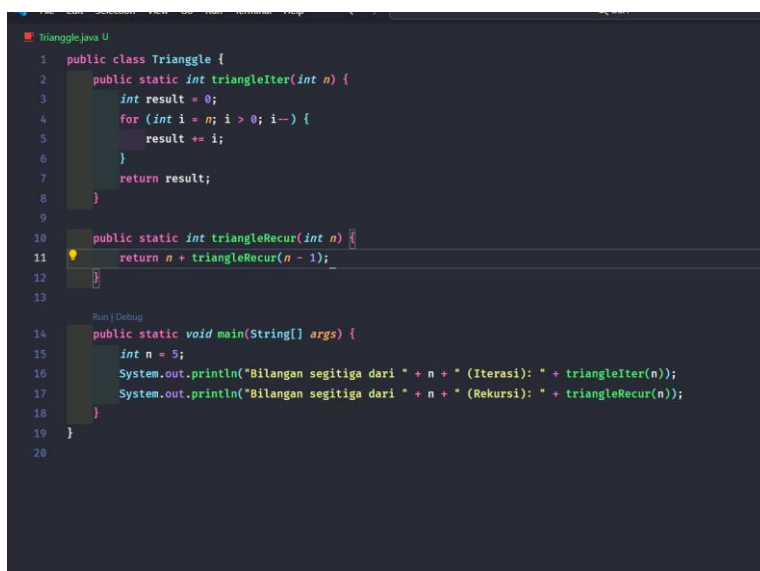
- Menggunakan loop (for/while).
- Prosesnya jelas langkah demi langkah dari awal hingga akhir.
- Lebih efisien dalam penggunaan memori karena tidak membuat tumpukan (stack) pemanggilan fungsi.

Rekursi:

- Menggunakan pemanggilan fungsi secara berulang.
- Lebih elegan dan mendekati definisi matematis bilangan segitiga.
- Namun, setiap pemanggilan fungsi disimpan di stack, sehingga bisa lebih boros memori jika nilai nnn besar.

Berdasarkan tahapan perhitungan tersebut, maka dapat dikatakan bahwa teknik rekursi menyelesaikan problem mulai dari yang lebih kecil / ~~lebih besar~~ (coret salah satu)

4. Pada method rekursi nomor 3 (triangleRecur) terdapat baris kode (if(n==1), return 1;, dan else) yang menunjukkan base case untuk masalah perhitungan bilangan segitiga. Pada kondisi tersebut, tidak dilakukan pemanggilan terhadap method itu sendiri. Lakukan percobaan dengan menghilangkan baris kode tersebut. Jalankan program dan lihat apa yang terjadi! Jelaskan!



```
1 public class Triangle {
2     public static int triangleIter(int n) {
3         int result = 0;
4         for (int i = n; i > 0; i--) {
5             result += i;
6         }
7         return result;
8     }
9
10    public static int triangleRecur(int n) {
11        return n + triangleRecur(n - 1);
12    }
13
14    public static void main(String[] args) {
15        int n = 5;
16        System.out.println("Bilangan segitiga dari " + n + " (Iterasi): " + triangleIter(n));
17        System.out.println("Bilangan segitiga dari " + n + " (Rekursi): " + triangleRecur(n));
18    }
19 }
20
```

```

at Trianggle.triangleRecur(Trianggle.java:11)
at Trianggle.triangleRecur(Trianggle.java:11)
at Trianggle.triangleRecur(Trianggle.java:11)
at Trianggle.triangleRecur(Trianggle.java:11)
at Trianggle.triangleRecur(Trianggle.java:11)
at Trianggle.triangleRecur(Trianggle.java:11)
at Trianggle.triangleRecur(Trianggle.java:11)
at Trianggle.triangleRecur(Trianggle.java:11)
at Trianggle.triangleRecur(Trianggle.java:11)
at Trianggle.triangleRecur(Trianggle.java:11)
at Trianggle.triangleRecur(Trianggle.java:11)
at Trianggle.triangleRecur(Trianggle.java:11)
at Trianggle.triangleRecur(Trianggle.java:11)
at Trianggle.triangleRecur(Trianggle.java:11)
at Trianggle.triangleRecur(Trianggle.java:11)
at Trianggle.triangleRecur(Trianggle.java:11)

```

Base case adalah syarat berhenti dalam rekursi. Tanpa base case, rekursi akan menjadi infinite recursion (rekursi tak terbatas). Oleh karena itu, dalam setiap algoritma rekursif selalu wajib ada base case yang memastikan pemanggilan fungsi akan berhenti di titik tertentu.

5. Dengan algoritma yang semisal dengan perhitungan bilangan segitiga, tuliskan listing program untuk menghitung nilai faktorial, baik menggunakan iterasi maupun menggunakan teknik rekursi! Jelaskan!

```

13
14  public static int factorialIter(int n) {
15      int result = 1;
16      for (int i = n; i > 0; i--) {
17          result *= i;
18      }
19      return result;
20  }
21
22  public static int factorialRecur(int n) {
23      if (n == 0 || n == 1) {
24          return 1;
25      } else {
26          return n * factorialRecur(n - 1);
27      }
28  }
29

```

```

addid@LAPTOP-F80MDN28 MINGW64 /a/Code/DSA (main)
● $ cd a:\\Code\\DSA ; /usr/bin/env A:\\Java\\jdk-2
2 -XX:+ShowCodeDetailsInExceptionMessages -cp C:\\
7\\redhat.java\\jdt_ws\\DSA_2f521fc6\\bin Trianggl
Faktorial dari 5 (Iterasi): 120
Faktorial dari 5 (Rekursi): 120

```

Iterasi (factorialIter)

- Gunakan perulangan for untuk mengalikan angka dari n turun sampai 1.
- Misal $n = 5$: $5 \times 4 \times 3 \times 2 \times 1 = 120$.

Rekursi (factorialRecur)

- Gunakan definisi matematis:

$$n! = n \times (n-1)!$$

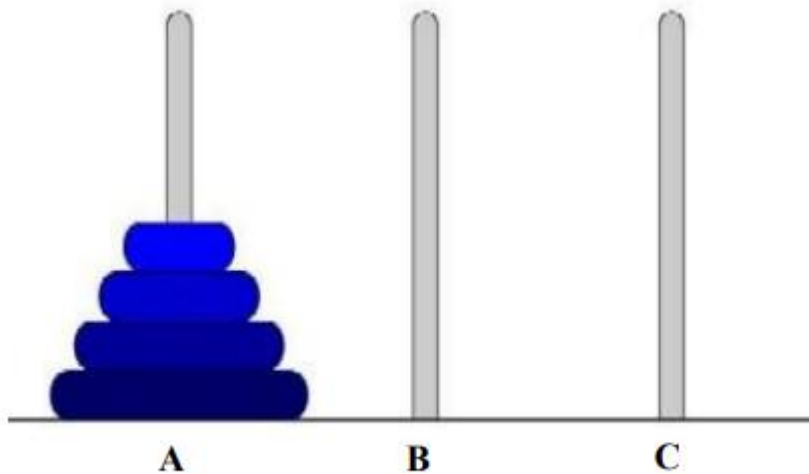
- Fungsi akan memanggil dirinya sendiri dengan parameter lebih kecil ($n-1$) hingga mencapai **base case** ($n == 0$ atau $n == 1$).
- Misal factorialRecur(5):
 $5 \times \text{factorialRecur}(4)$
 $\rightarrow 5 \times 4 \times \text{factorialRecur}(3)$
 $\rightarrow 5 \times 4 \times 3 \times \text{factorialRecur}(2)$
 $\rightarrow 5 \times 4 \times 3 \times 2 \times \text{factorialRecur}(1)$
 $\rightarrow 5 \times 4 \times 3 \times 2 \times 1 = 120$.

6. Menara Hanoi Menara Hanoi ini adalah sebuah permainan matematis atau teka-teki. Permainan ini terdiri dari tiga tiang dan sejumlah cakram dengan ukuran berbeda-beda yang bisa dimasukkan ke tiang mana saja. Permainan dimulai dengan cakram-cakram yang tertumpuk rapi berurutan berdasarkan ukurannya dalam salah satu tiang, cakram terkecil diletakkan teratas, sehingga membentuk kerucut.

Tujuan dari teka-teki ini adalah untuk memindahkan seluruh tumpukan ke tiang yang lain, mengikuti aturan berikut:

- Hanya satu cakram yang boleh dipindahkan dalam satu waktu.
- Setiap perpindahan berupa pengambilan cakram teratas dari satu tiang dan memasukkannya ke tiang lain, di atas cakram lain yang mungkin sudah ada di tiang tersebut.

- Tidak boleh meletakkan cakram di atas cakram lain yang lebih kecil.



Gambar 5.2 Menara Hanoi

Hitung jumlah perpindahan dan tuliskan langkah untuk memindahkan 3, 4 dan 5 buah cakram ke tiang C (tujuan)!

Permainan Menara Hanoi dapat diselesaikan dengan algoritma rekursi, di mana setiap cakram dipindahkan dari tiang awal menuju tiang tujuan dengan memanfaatkan tiang ketiga sebagai perantara. Jumlah perpindahan minimum untuk n cakram ditentukan oleh rumus $2^n - 1$. Berdasarkan rumus ini, diperlukan 7 langkah untuk 3 cakram, 15 langkah untuk 4 cakram, dan 31 langkah untuk 5 cakram.

Jumlah langkah minimum:

- 3 cakram = 7 langkah
- 4 cakram = 15 langkah
- 5 cakram = 31 langkah

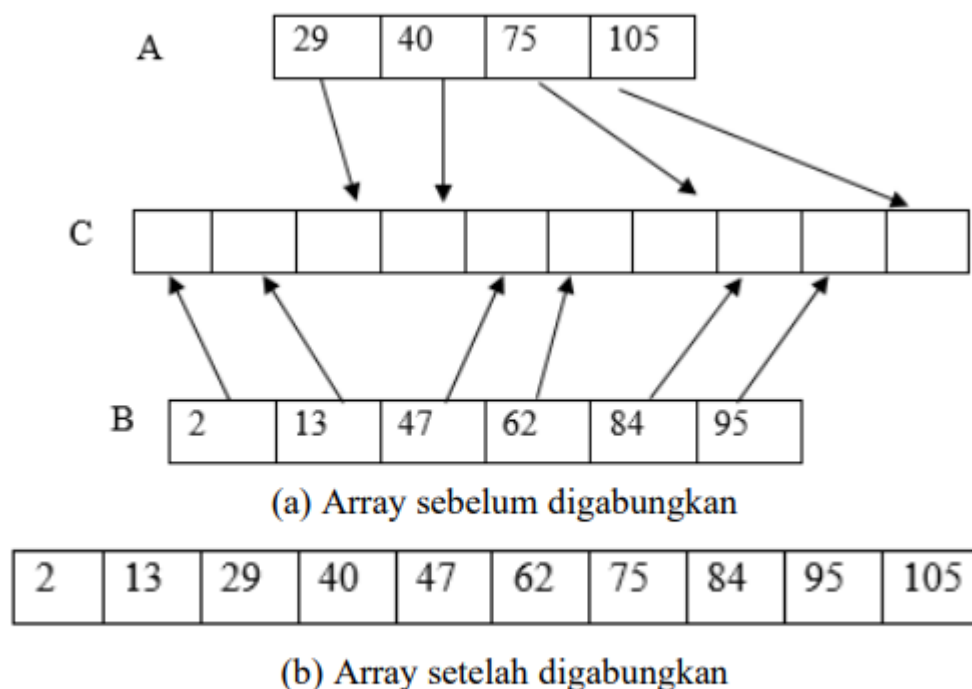
Contoh langkah untuk 3 cakram:

1. Cakram 1 : A \rightarrow C
2. Cakram 2 : A \rightarrow B
3. Cakram 1 : C \rightarrow B
4. Cakram 3 : A \rightarrow C
5. Cakram 1 : B \rightarrow A
6. Cakram 2 : B \rightarrow C
7. Cakram 1 : A \rightarrow C

Untuk 4 cakram, dibutuhkan 15 langkah. Polanya tetap mengikuti prinsip rekursif, dimulai dari pemindahan Cakram 1 : $A \rightarrow B$, dilanjutkan Cakram 2 : $A \rightarrow C$, dan seterusnya hingga semua cakram tersusun di tiang C.

Sedangkan untuk 5 cakram, jumlah minimum perpindahan adalah 31 langkah, dengan langkah awal Cakram 1 : $A \rightarrow C$, kemudian Cakram 2 : $A \rightarrow B$, dan dilanjutkan sesuai pola hingga seluruh cakram berpindah ke tiang tujuan (C).

7. Menggabungkan dua ordered array Penggabungan (Merge) adalah teknik yang menjadi konsep algoritma pengurutan “Merge Sort”. Berikut ini merupakan ilustrasi merge (penggabungan) dua array, yaitu array A dan array B yang masing-masing sudah terurut. Array A dan B digabungkan dan disimpan dalam Array C.



Gambar 5.3 Penggabungan dua *ordered array*

Tuliskan langkah-langkah penggabungan dua Array sesuai dengan ilustrasi diatas!

roses **penggabungan dua array terurut** (A dan B) ke dalam array C dilakukan dengan langkah-langkah berikut:

- Bandingkan elemen pertama dari A (29) dengan elemen pertama dari B (2). Karena 2 lebih kecil, masukkan 2 ke C.
- Lanjutkan dengan membandingkan 29 (A) dengan 13 (B). Karena 13 lebih kecil, masukkan 13 ke C.
- Selanjutnya, bandingkan 29 (A) dengan elemen berikutnya (karena B sudah habis di posisi tersebut), sehingga 29 dimasukkan ke C.

- Kemudian 40 (A) dibandingkan dengan 47 (B). Karena 40 lebih kecil, maka 40 masuk ke C.
- Bandingkan 75 (A) dengan 47 (B). Karena 47 lebih kecil, masukkan 47 ke C.
- Bandingkan 75 (A) dengan 62 (B). Karena 62 lebih kecil, masukkan 62 ke C.
- Bandingkan 75 (A) dengan 84 (B). Karena 75 lebih kecil, masukkan 75 ke C.
- Bandingkan 105 (A) dengan 84 (B). Karena 84 lebih kecil, masukkan 84 ke C.
- Bandingkan 105 (A) dengan 95 (B). Karena 95 lebih kecil, masukkan 95 ke C.
- Terakhir, sisa elemen dari A yaitu 105 dimasukkan ke C.

Hasil akhir array C: [2, 13, 29, 40, 47, 62, 75, 84, 95, 105]

```

class Arrays {

    private int arr[];
    private int nElemen;

    public Arrays(int size) {
        arr = new int[size];
        nElemen = 0;
    }

```

```

    public void insert(int value) {
        arr[nElemen] = value;
        nElemen++;
    }

    public void display() {
        for (int i = 0; i < nElemen; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println("");
    }

    public void mergeSort() {
        int[] workSpace = new int[nElemen];
        recMergeSort(workSpace, 0, nElemen - 1);
    }

```

```

    public void recMergeSort(int[] workSpace,
        int lowerBound, int upperBound) {

        if (lowerBound == upperBound) {
            return;
        } else {
            int mid = (lowerBound + upperBound)
                / 2;
            recMergeSort(workSpace, lowerBound,
                mid);
            recMergeSort(workSpace, mid + 1,
                upperBound);
            merge(workSpace, lowerBound,
                mid + 1, upperBound);
        }
    }

```

```

    public void merge(int[] workSpace,
        int lowIndex, int highIndex,
        int upperBound) {

        int j = 0;
        int lowerBound = lowIndex;
        int mid = highIndex - 1;
        int nItem = upperBound - lowerBound + 1;

```



```

20
27     public void recMergeSort(int[] workSpace,
28                             int lowerBound, int upperBound) {
29         if (lowerBound == upperBound) {
30             return;
31         } else {
32             int mid = (lowerBound + upperBound)
33                 / 2;
34             recMergeSort(workSpace, lowerBound,
35                         mid);
36             recMergeSort(workSpace, mid + 1,
37                         upperBound);
38             merge(workSpace, lowerBound,
39                 mid + 1, upperBound);
40         }
41     }

```

```

42
43     public void merge(int[] workSpace,
44                     int lowIndex, int highIndex,
45                     int upperBound) {
46
47         int j = 0;
48         int lowerBound = lowIndex;
49         int mid = highIndex - 1;
50         int nItem = upperBound - lowerBound + 1;
51         while (lowIndex <= mid
52             && highIndex <= upperBound) {
53             if (arr[lowIndex] < arr[highIndex]) {
54                 workSpace[j++] = arr[lowIndex++];
55             } else {
56                 workSpace[j++] = arr[highIndex++];
57             }
58         }
59
60         while (lowIndex <= mid) {
61             workSpace[j++] = arr[lowIndex++];
62         }
63         while (highIndex <= upperBound) {
64             workSpace[j++] = arr[highIndex++];
65         }
66         for (j = 0; j < nItem; j++) {
67             arr[lowerBound + j] = workSpace[j];
68         }
69     }
70
71 }

```

```

73  public class MergeMain {
    Run | Debug
74  public static void main(String[] args) {
75      Arrays arr = new Arrays(size:10);
76      arr.insert(value:45);
77      arr.insert(value:12);
78      arr.insert(value:89);
79      arr.insert(value:33);
80      arr.insert(value:76);
81      arr.insert(value:2);
82      arr.insert(value:90);
83      arr.insert(value:54);
84      arr.insert(value:23);
85      arr.insert(value:67);
86
87      System.out.println(x:"Data sebelum diurutkan:");
88      arr.display();
89      arr.mergeSort();
90
91      System.out.println(x:"Data setelah diurutkan:");
92      arr.display();
93  }
94  }

```

```

addid@LAPTOP-F80MDN28 MINGW04 /a/Code/DSA (main)
$ /usr/bin/env A:\\Java\\jdk-22\\bin\\java.exe -X
2dce0b54ebb13767\\redhat.java\\jdt_ws\\DSA_2f521fc
Data sebelum diurutkan:
45 12 89 33 76 2 90 54 23 67
Data setelah diurutkan:
2 12 23 33 45 54 67 76 89 90

```

9. Berdasarkan data yang Anda gunakan pada soal nomor 8, tuliskan tiap tahap pengurutan menggunakan mergeSort! Jelaskan!

Prinsip merge sort:

1. Bagi array menjadi 2 bagian sampai tidak bisa dibagi lagi (satu elemen).
2. Gabungkan kembali dengan cara membandingkan elemen sehingga urut.

Tahap 1 – Pembagian (Divide)

Array [45, 12, 89, 33, 76, 2, 90, 54, 23, 67] dibagi menjadi 2:

- Kiri: [45, 12, 89, 33, 76]
- Kanan: [2, 90, 54, 23, 67]

Tahap 2 – Pembagian lebih lanjut

Kiri [45, 12, 89, 33, 76] → dibagi:

- [45, 12, 89]
- [33, 76]

Kanan [2, 90, 54, 23, 67] → dibagi:

- [2, 90, 54]
- [23, 67]

Tahap 3 – Pecah terus sampai 1 elemen

- [45, 12, 89] → [45, 12] dan [89] → lalu [45] dan [12]
- [33, 76] → [33] dan [76]
- [2, 90, 54] → [2, 90] dan [54] → lalu [2] dan [90]
- [23, 67] → [23] dan [67]

Sekarang semua sudah 1 elemen.

Tahap 4 – Penggabungan (Conquer)

Kita mulai gabung urut:

1. [45] + [12] → [12, 45]
2. [12, 45] + [89] → [12, 45, 89]
3. [33] + [76] → [33, 76]
4. Gabungkan [12, 45, 89] + [33, 76] → [12, 33, 45, 76, 89]

Kiri sudah jadi: [12, 33, 45, 76, 89]

5. [2] + [90] → [2, 90]
6. [2, 90] + [54] → [2, 54, 90]
7. [23] + [67] → [23, 67]
8. Gabungkan [2, 54, 90] + [23, 67] → [2, 23, 54, 67, 90]

Kanan sudah jadi: [2, 23, 54, 67, 90]

Tahap 5 – Gabung hasil kiri dan kanan

Gabungkan [12, 33, 45, 76, 89] dengan [2, 23, 54, 67, 90]:

- Bandingkan 12 dan 2 → pilih 2
- Bandingkan 12 dan 23 → pilih 12
- Bandingkan 33 dan 23 → pilih 23
- Bandingkan 33 dan 54 → pilih 33
- Bandingkan 45 dan 54 → pilih 45
- Bandingkan 76 dan 54 → pilih 54
- Bandingkan 76 dan 67 → pilih 67
- Bandingkan 76 dan 90 → pilih 76
- Bandingkan 89 dan 90 → pilih 89
- Sisa 90 → ambil 90

Hasil akhir:

[2, 12, 23, 33, 45, 54, 67, 76, 89, 90]

Penjelasan

- **MergeSort bekerja rekursif:** memecah array hingga satu elemen.
- **Setelah itu elemen digabung** dengan cara membandingkan elemen terkecil dari dua subarray.
- Proses ini memastikan hasil penggabungan selalu **urut**.
- Kompleksitas waktu: **$O(n \log n)$** .